School of Mechanical & Manufacturing Engineering (SMME),

National University of Science and Technology (NUST),

Sector H-12, Islamabad

Program:BE Aerospace Engineering Section:AE-01

Session:Fall 2023 Semester:1st

Course Title:Fundamentals of Programming,CS-109

" **End Semester Project"**

"Arrays and functions in C++ programming"

Name : Aisha Iqbal

CMS : 456928

Introduction:

The provided C++ code is a simple implementation of a Tic Tac Toe game where the player competes against the computer. The program utilizes the minimax algorithm to make optimal moves, ensuring a challenging gameplay experience for the player. This report will discuss the structure of the code, its functionality, and provide sample outputs. Additionally, potential future improvements and applications will be considered.

Code:

```
#include <iostream>

using namespace std;

char board[3][3] = {{' ', ' ', ' '}, {' ', ' ', ' '}, {' ', ' ', ' '}};

int findBestMoveRow;
int findBestMoveCol;

void displayBoard() {
	for (int i = 0; i < 3; ++i) {
	for (int j = 0; j < 3; ++j) {
		cout << board[i][j] << ' ';
	}
	cout << endl;
	}
	cout << endl;
}

bool isBoardFull() {
	for (int i = 0; i < 3; ++i)
	for (int j = 0; j < 3; ++j)
		if (board[i][j] == ' ')
		return false;
	return true;
}

bool isWinner(char player) {
	for (int i = 0; i < 3; ++i)
	if ((board[i][0] == player && board[i][1] == player && board[i][2] ==
player) ||
		(board[0][i] == player && board[1][i] == player && board[2][i] ==
player))
		return true;
```

```
	if ((board[0][0] == player && board[1][1] == player && board[2][2] ==
player) ||
	(board[0][2] == player && board[1][1] == player && board[2][0] == player))
	return true;

	return false;
}

bool isValidMove(int row, int col) {
	if (row >= 0 && row < 3 && col >= 0 && col < 3 && board[row][col] == ' ') {
	return true;
	} else {
	return false;
	}
}

void makeMove(int row, int col, char player) {
	board[row][col] = player;
}

int minimax(bool maximizing) {
	// Check if 'X' wins, return -1
	if (isWinner('X')) {
	return -1;
	}
	// Check if 'O' wins, return 1
	else if (isWinner('O')) {
	return 1;
	}
	// Check if it's a draw, return 0
	else if (isBoardFull()) {
	return 0;
	}

	// Set the initial best score based on maximizing or minimizing
	int bestScore;
	if (maximizing) {
	bestScore = -1000000000;  // A large negative value for 'X'
	}
	 else {
	bestScore = 1000000000;   // A large positive value for 'O'
	}

	// Loop through each cell on the board
	for (int i = 0; i < 3; ++i) {
	for (int j = 0; j < 3; ++j) {
		// Check if the current cell is empty
```

```
            if (board[i][j] == ' ') {
            // Make a move for the current player ('X' or 'O')
            if (maximizing) {
                 board[i][j] = 'O';
             }
             else {
                 board[i][j] = 'X';
             }


            // Recursively call minimax for the next state with the opposite
player
            int score = minimax(!maximizing);

            // Undo the move (backtrack)
            board[i][j] = ' ';

            // Update the best score based on maximizing or minimizing
            if (maximizing) {
                  if (score > bestScore) {
                        bestScore = score;
                  }
            } else {
                  if (score < bestScore) {
                        bestScore = score;
                  }
            }
            }
      }
      }

      // Return the best score for the current state
      return bestScore;
}


void findBestMove() {
      int bestScore = -1000000000;

      for (int i = 0; i < 3; ++i)
      for (int j = 0; j < 3; ++j)
            if (board[i][j] == ' ') {
            board[i][j] = 'O';

            int score = minimax(false);

            board[i][j] = ' ';
```

```
            if (score > bestScore) {
                  bestScore = score;
                  findBestMoveRow = i;
                  findBestMoveCol = j;
            }
            }
}

int main() {
      int row, col;
      findBestMoveRow = -1;
      findBestMoveCol = -1;

      while (true) {
      displayBoard();

      cout << "Enter your move (row and column): ";
      cin >> row >> col;

      if (isValidMove(row, col)) {
            makeMove(row, col, 'X');

            if (isWinner('X')) {
            displayBoard();
            cout << "Congratulations! You win!" << endl;
            break;
            }

            if (isBoardFull()) {
            displayBoard();
            cout << "It's a draw!" << endl;
            break;
            }

            findBestMove();
            makeMove(findBestMoveRow, findBestMoveCol, 'O');

            if (isWinner('O')) {
            displayBoard();
            cout << "You lose! Better luck next time." << endl;
            break;
            }
      } else {
            cout << "Invalid move. Try again." << endl;
      }
      }
```

```
        return 0;
}
```

Explanation:

The code begins with the initialization of the Tic Tac Toe board, which is a 3x3 grid. Functions such as displayBoard, isBoardFull, isWinner, isValidMove, and makeMove handle

the game's core logic, including displaying the board, checking for a full board or a winning condition, validating player moves, and updating the board after a move.

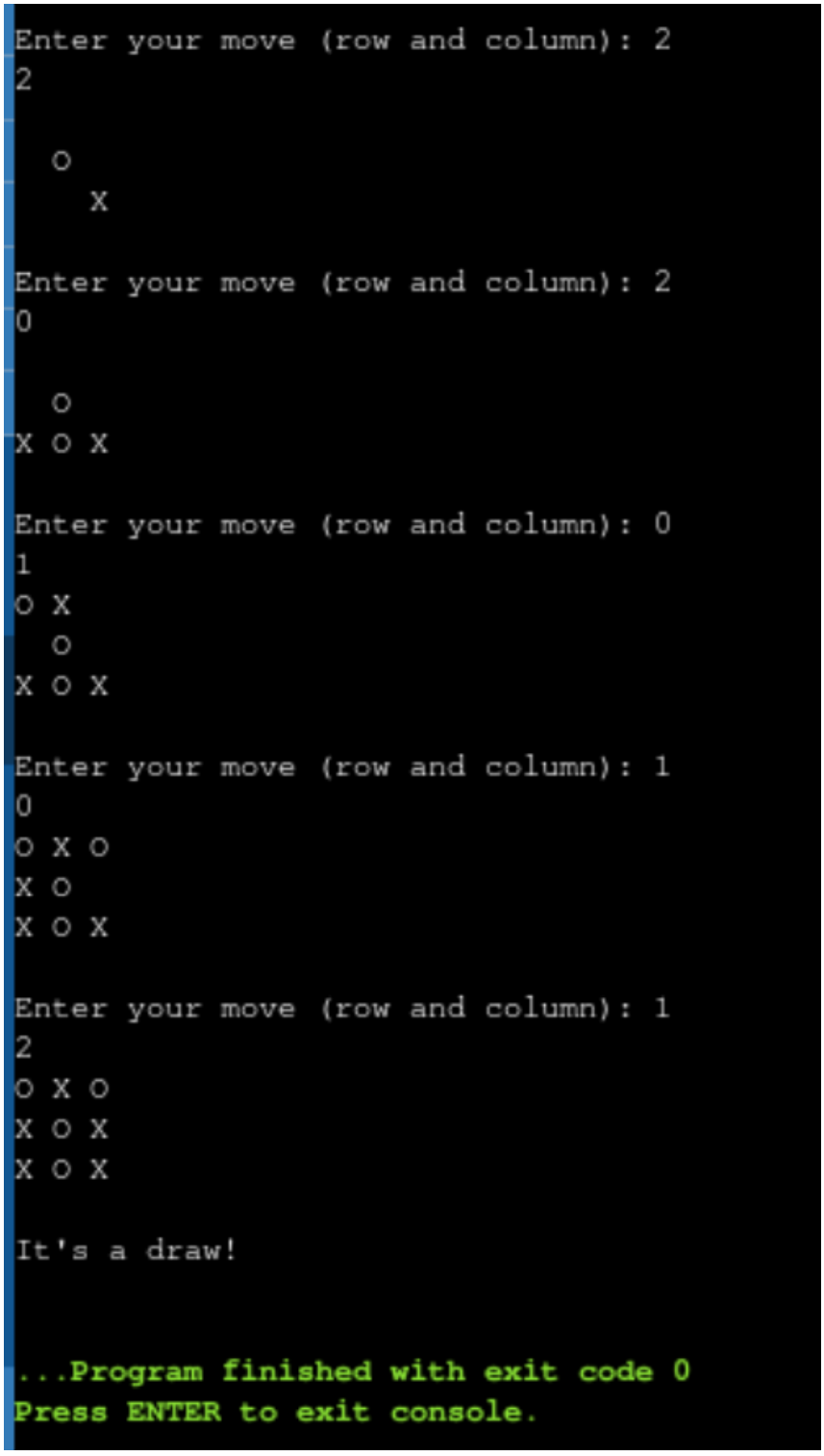
The heart of the AI functionality lies in the minimax function, which recursively explores possible moves, assigns scores to different game outcomes, and ultimately selects the move that leads to the optimal result for the AI player.

The findBestMove function calls the minimax algorithm for the AI player and determines the best move to be played. The game loop in the main function allows the human player to input moves, and the AI responds with its calculated optimal moves.

Outputs:

```
Enter your move (row and column): 0
0
X
  O


Enter your move (row and column): 0
1
X X O
  O


Enter your move (row and column): 0
1
Invalid move. Try again.
X X O
  O


Enter your move (row and column): 1
2
X X O
  O X
O

You lose! Better luck next time.


...Program finished with exit code 0
Press ENTER to exit console.
```

In this case, we made a mistake and the AI took advantage of that and won the game.



In this case, you can easily see that the AI blocked any potential wins by the human layer. If the human plays optimally, then we can only force a draw. The AI will always either win the game or a draw will ensue due to the nature of tic tac toe.

Future Work and Applications:

1. Minimax can also be implemented on other complex games like Chess where it is incredibly helpful for practice and training. Super computing engines made to play chess use algorithms like minimax to play against chess world champions and defeat them.
2. User Interface(UI):Develop a graphical user interface to make the game more visually appealing and user-friendly.
3. Multiplayer Support:Extend The Game To Support Multiplayer Functionality, allowing users to play against each other.
4. Enhanced Game Features:Introduce Features Like variable board sizes,different player symbols, or customizable difficulty levels to diversify the gaming experience.

Conclusion:

In conclusion, the implemented Tic Tac Toe game demonstrates a fundamental application of the minimax algorithm for creating an AI opponent. The code provides an engaging gameplay experience for users and serves as a foundation for further enhancements. The simplicity of the code allows for easy understanding and modification, making it suitable for learning and future development.